Large Physical Address Extensions Specification

ARM Architecture Group

Document number:	Derived from: **PRD03-GENC-008469 15.0**
Date of Issue:	22 October 2010
Author:	ARM limited
Authorised by:	

Note

The term ARM is also used to refer to versions of the ARM architecture, for example ARMv6 refers to version 6 of the ARM architecture. The context makes it clear when the term is used in this way.

Contents

1 ABOUT THIS DOCUMENT

The material in this document will be incorporated in the next release of the ARMv7-AR Architecture Reference manual (Revision C).

It is at a Beta specification state. Please communicate any errata or issues to errata@arm.com.

1.1 Change history

This document is derived from an ARM internal document, PRD03-GENC-008469 15.0, and is supplied as PRELIMINARY INFORMATION about the ARMv7-A Architecture Large Physical Address Extension announced in August 2010.

1.2 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
1	ARM DDI 0406 B	ARM Limited	ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition
2	PRD03-GENC-008353	ARM Limited	Virtualization Extensions Architecture

2 SCOPE

This document describes the Large Physical Address Extensions for the ARM architecture, and the translation table format used as part of the Virtualization Extensions.

3 INTRODUCTION

A new translation table system is added to the ARM architecture for two reasons:

1. A need to support a physical address of >32 bits, in order to support more RAM mapped into the virtual memory system at a page granularity.
2. A need to introduce support for a second stage of translation as part of the Virtualization Extensions described in [2]

This specification describes the new translation system which can be selected as an alternative to the VMSA translation system described in [1], to support the large translation address. This constitutes the first stage address translation. The same new translation system is used as the first (and only) stage of address translation for memory accesses performed in Hyp mode, see [2] for a definition of this mode.

This specification also describes the translation system for the second stage of memory translation for the Virtualization Extensions.

The details of the second stage of translation held within this document only apply for the Virtualization Extensions, and are for use by an agent that uses Hyp mode. For operating systems that have no intention of using Hyp mode for their own purposes, the second stage of translation is irrelevant, the IPA should be assumed to be the same as the PA.

4 REVISED TRANSLATION TABLE FORMAT

4.1 Overview

4.1.1 Terminology

In a system with hardware support for two stages of translation for virtualization (as discussed in [2]), the terms “virtual” and “physical” address can be a little misleading when discussing translation tables. For the purpose of this document, the purpose of a stage of translation is to translate the *Input* address to *Output* address.

Thus:

- for a first stage of translation, the Input address would be the Virtual Address and the Output address would be the Intermediate Physical Address.
- for a second stage of translation, the Input address would be the Intermediate Physical Address and the Output address would be the Physical Address.
- for a system with only one stage of translation, the Input address would be the Virtual Address and the Output address would be the Physical Address.

The memory translation system is based on tables which hold translation table entries. The lowest address of a table is described as the table’s *Base Address.*

4.2 Description of the new 3-level system

4.2.1 Overview

This specification introduces a new translation table format for a given stage of translation that is based around:

1. Up to 3 levels of translation table walk
2. Up to 40 bits of Output address space
3. Up to 40 bits of Input address space where needed
4. 4Kbyte tables for all levels of table, unless truncated by the size of the input address space
5. 4Kbyte granularity of memory assignment

The actual number of levels of translation table walk is controlled by the Input address space being accessed.

Each entry in the translation tables is a 64-bit descriptor.

For the first stage of translation used in the Non-secure modes other than Hyp mode, and also used in the Secure modes, there are two translation table base registers to allow different translation tables to be used for Input addresses at the top and the bottom of the address space. This extends the TTBR0/1 approach in the ARMv7 VMSA system described in [1].

For memory accesses performed in Hyp mode, and for the second stage of translation used by the Virtualization Extensions for Non-secure modes other than Hyp mode, a single translation table base register is used.

The number of Output address bits that are defined in the translation tables are limited to 40 bits at this time.

Note: While this document describes the facilities for 3 levels of table, the system reserves space and uses naming that reflects an ability to extend the same system to 4 levels with 48 bits of Input address and potentially further to 5 or more levels.

4.2.2 Details of the 3-level system

In the 3-level system, 3 levels of translation table are used. In keeping with the 2-level system, the level numbers increase as the granularity of the translation gets finer.

Level 1 – this translation table covers a range of up to 512 GBytes (2^{39} bytes) and consists of up to 512 entries at 64 bits in size. Each entry covers an address range of 1GBytes (2^{30} bytes)

Level 2 – this translation table covers a range of up to 1GBytes and consists of up to 512 entries at 64 bits in size. Each entry covers an address range of 2MBytes (2^{21} bytes)

Level 3 – this translation table covers a range of 2MBytes and consists of 512 entries at 64 bits in size. Each entry covers an address range of 4KBytes (2^{12} bytes)

The canonical system for each level of translation table is that it offers 4 basic forms of entry:

1. An Invalid mapping, access to which gives a translation fault.
2. At Level 1 and Level 2, a “block” entry, providing a “flat” mapping of the address range of the entry.

   At Level 3, a “page” entry, with a different format.
3. A “table” entry, providing the base address of the next level of translation table
4. A reserved mapping, which might be used in future, but which today is treated as an Invalid mapping.

In each case the mapping type is determined by the bottom 2 bits of the descriptor.

Note: For the second stage of translation, a mechanism is provided to combine two Level1 tables to service a 40-bit input address – see section 4.2.5

4.2.3 Attribute and Permission fields in the translation tables

The attribute and permission fields for the stage 1 translation in the new format have the same meaning as the equivalent attributes fields described in the ARMv7-A VMSA [1]. For stage 1 translation, the tables behave as if SCTLR.TRE==1, and SCTLR.AFE==1 and these bits are UNK/SBOP when the new format is enabled (see section 4.3.1.2)

The attribute and permissions fields for the stage 2 translation use the field formats described in [2].

The NS-bit in the translation tables in all cases applies to the translation described in that level of table. This differs from the approach used for translations held as pages in the ARMv7-A VMSA description.

The following table shows the format of these descriptors for a single 4KByte table, where IA is the Input Address:

Table level	**Addressed by**	**Entry Address range**	**Descriptor Format**
Level 1	BaseAddress[39:x],IA[y:30],000 x = 5-T{0,1}SZ y = x+26 For the Stage1 translation 0 <= T{0,1}SZ <= 1 For the Stage2 translation -8 <= T0SZ <= 1	1GByte	Bit[0] : 1: Valid; 0: Invalid (Translation Fault) Bit[1]: 0: Block, 1: Table Block: Bits[63:52]: Upper Block Attributes Block[51:40]: SBZ Bits[39:30] : Output Address[39:30] Bits[29:12]: SBZ Bits[11:2]: Lower Block Attributes If Table: Bit[63]: NSTable (stage1 only) Bit[62:61]: APTable (stage1 only) Bit[60]: XNTable (stage 1 only) Bit[59]: PXNTable (stage 1 only) Bit[63:59]: SBZ (stage2 only) Bits[58:52]: IGNORED Bits[51:40]: SBZ Bits[39:12]: L2 Table Output Address Bits[11:2]: IGNORED
Level 2	BaseAddress[39:x],IA[y:21],000 For the Stage1 translation	2Mbyte	Bit[0] : 1: Valid; 0: Invalid (Translation Fault) Bit[1]: 0: Block, 1: Table

	x = 14-T{0,1}SZ if T{0,1}SZ >= 2 x = 12 otherwise y = x+17 For the Stage2 translation If SL0 = 1 then x=12 If SL0 = 0 then : x=14-T0SZ T0SZ >= -2 y = x+17		Block: Bits[63:52]: Upper Block Attributes Bits[51:40]: SBZ Bits[39:21] : Output Address[39:21] Bits[20:12]: SBZ Bits[11:2]: Lower Block Attributes If Table: Bit[63]: NSTable (stage1 only) Bit[62:61]: APTable (stage1 only) Bit[60]: XNTable (stage1 only) Bit[59]: PXNTable (stage 1 only) Bit[63:59]: SBZ (stage2 only) Bits[58:52]: IGNORED Bits[51:40]: SBZ Bits[39:12]: L3 Table Output Address Bits[11:2]: IGNORED
Level 3	BaseAddress[39:12],IA[20:12],000	4KByte	Bit[0] : 1: Valid; 0: Invalid (Translation Fault) Bit[1] : '1' (when '0': Translation Fault) Bits[63:52]: Upper Page Attributes Bits[51:40]: SBZ Bits[39:12] : Output Address[39:12] Bits[11:2]: Lower Page Attributes

For all descriptors, if the entry is Invalid (bit 0 is clear) then all other fields are IGN by hardware.

Lower Page attributes **Lower Block attributes** **Bit position**	**Meaning for a Stage 1 translation**	**Meaning for a Stage 2 translation**
11	nG	SBZ
10	AF: Access Flag	AF:Access Flag
9:8	SH[1:0]: Shareability	SH[1:0]: Shareability
7:6	AP[2:1]: Access Permission, in the VMSAv7 when Access Flag enabled	HAP[1:0]: Stage 2 Access Permission described in [2]
5	NS	Stage 2 Memory Type, MemAttr[3:0], Described in [2]
4 :2	AttrIndx[2:0]	

Upper Page attributes Upper Block attributes Bit position	Meaning for a Stage 1 translation	Meaning for a Stage 2 translation
63:59	IGNORED	IGNORED
58:55	Reserved for s/w use	Reserved for s/w use
54	XN	XN
53	PXN	SBZ
52	16 entry contiguous hint	16 entry contiguous hint

Fields marked as IGNORED are guaranteed to be ignored by the hardware and can take any value.

Note: The IGNORED fields are allocated to facilitate the use of the Recursive Page Table property described in section **Error! Reference source not found.**. If this property is not being exploited by software, the IGNORED fields can be used for other software use.

The Shareability attributes for Normal memory are encoded using the SH[1:0] field as follows:

SH[1]	SH[0]	Normal Memory
0	0	Non-Shareable
0	1	UNPREDICTABLE
1	0	Outer Shareable
1	1	Inner Shareable

If the attributes for a memory location are Normal Inner Non-Cacheable, Outer Non-Cacheable, then if the shareability attributes must be Outer Shareable, or else the Shareability is UNPREDICTABLE when using the LPAE translation table format.

For implementations that implement the Large Physical Address Extensions, when the LPAE translation table format is not being used, for the Normal Inner Non-Cacheable, Outer Non-Cacheable memory type it is IMPLEMENTATION DEFINED whether:

- the address based cache maintenance operations apply to the set of cores defined by the shareability domain of the address or apply to all cores within the same Outer Sharable domain
- load-exclusive and store-exclusive instructions make use of the Global monitor as well as the local monitor if the locations are marked as Non-shareable
- the value of the SH and NOS field in the PAR as a result of an address translation operation on a memory location using that memory type takes the shareability domain of the address or returns Outer Shareable.

4.2.3.1 Meaning of SH field for the Strongly-ordered and Device memory types

The translation table format described in the Large Physical Address Extensions does not provide a distinction between Shareable Device and Non-Shareable Device, as the SH field is ignored for the Device and Strongly-ordered memory types.

The architectural definition of Shareability has 5 characteristics associated with it:

- The coherency of the memory type; this is most important for cacheable memory types, where the shareability determines the need for and extent of the use of cache coherency protocols. For Strongly-

ordered or Device memory types, the fact that it is not permissible to cache those memory locations means that the shareability has no significance for coherency for the Strongly-Ordered or Device memory types.

- The relevance of other agents for the load-exclusive and store-exclusive instructions; this has no relevance for Strongly-ordered or Device memory types as the use of load-exclusive or store-exclusive instructions is UNPREDICTABLE with these memory types
- The requirement for the address based cache maintenance operations to be applied to the set of cores defined by the shareability domain of the address; For implementations that include the Large Physical Address Extensions, it is required that if the address specified is Strongly-Ordered or Device, then the maintenance operation is applied to all cores in the Outer Shareable domain.
- Shareable Device memory transactions to a single peripheral are permitted, but are not required, to be re-ordered with respect to Non-Shareable Device transactions, where the definition of a single peripheral is of an IMPLEMENTATION DEFINED size of not less than 4Kbytes. As Device transactions to different peripherals are always permitted to be reordered regardless of Shareability attributes, the permissibility of re-ordering Shareable Device and Non-shareable Device memory types to the same peripheral is of little practical benefit and is not supported for implementations that include the Large Physical Address Extensions.
- The PAR indicates shareability information for Device memory as a result of address translation operations. Since this information has no meaning for implementations that include the Large Physical Address Extensions, it is required to return a fixed value for Strongly-ordered and Device memory.

For implementations that implement the Large Physical Address Extensions, when the new translation table format is not being used:

- The address based cache maintenance operations apply to all cores within the same Outer Shareable domain for Strongly-ordered and Device memory, regardless of any shareability attributes that might be applied to these memory types, including the NOS bits held in the NMRR.
- Any Device memory transactions to a single peripheral, regardless of any shareability attributes that might be applied to that memory type, including the NOS bits held in the NMRR, are not permitted to be reordered
- Where Strongly-ordered or Device memory is being reported in the PAR as a result of an address translation operation, the SH field takes the value 1 and the NOS field takes the value 0 regardless of any shareability attributes that might be applied to those memory types, including the NOS bits held in the NMRR.

4.2.4 Performing the translation table walk – First (or only) stage of translation

This translation table system can be used for a first (or only) stage of translation with a 32-bit Input address as an alternative to the translation table systems present in ARMv7 VMSA for memory accesses made in the Non-secure modes other than Hyp mode, and also used in the Secure modes. A bit in the corresponding TTBCR determines whether the ARMv7 VMSA system or the new translation system is used.

For memory accesses made in Hyp mode, this system is the only translation system that is used.

This translation system allows access to a larger than 32 bit Output address space, with the translation tables providing a 40bit Output address range. The first stage of translation is based on a two or three level table, starting at level 2 or level 1 as appropriate: the number of levels is determined by the size of the Input address range configured in the translation control register (using the TTBCR.T0SZ and TTBCR.T1SZ fields or the HTCR.T0SZ field):

- If the Input address range is less than or equal to 2^{30} bytes, then the starting level is level 2, requiring two levels of tables to provide 4KByte granularity mappings.
- If the Input address range is greater that 2^{30} bytes, then the starting level is level 1, requiring three levels of tables to provide 4KByte granularity mappings.

In keeping with the ARMv7 VMSA system, two translation table base registers (TTBR0 and TTBR1) are provided except in Hyp mode, though these are expanded to become 64-bit descriptors.

4.2.4.1 Translations not in Hyp Mode

The LPAE translation system when not in Hyp Mode presents two translation table base registers, TTBR0 and TTBR1, which give the table base address for virtual addresses at the bottom and top of virtual address range respectively. The size of these regions is independently configurable.

The mechanism for selecting which translation table base register is extended from the system used in the ARMv7 VMSA to allow the TTBR0 described space to be larger than the TTBR1 space. In the revised scheme, the boundary between the TTBR0 region and the TTBR1 region can be configured using the TTBCR.T0SZ and TTBCR.T1SZ fields to be any of the following positions:

- No boundary – all entries are translated using the TTBR0 region
    - This is encoded by having TTBCR.T0SZ =0 and TTBCR.T1SZ=0
- 2^N (31 >= N >= 25) - this is consistent with the ARMv7 VMSA capability
    - This is encoded by having TTBCR.T0SZ = (32-N) and TTBCR.T1SZ = 0
- $2^{32} – 2^N$ (31 >= N >= 25) - this is a new capability
    - This is encoded by having TTBCR.T0SZ = 0 and TTBCR.T1SZ = (32-N)

Furthermore the size of the TTBR0 and TTBR1 regions can be independently controlled to allow them to be separated – this occurs when non-zero values are programmed into both TTBCR.T0SZ and TTBCR.T1SZ. Any attempt to access an Input address which is not mapped by either region will result in a Translation Fault.

The maximum Input address translated by the first stage of translation can be configured to be :

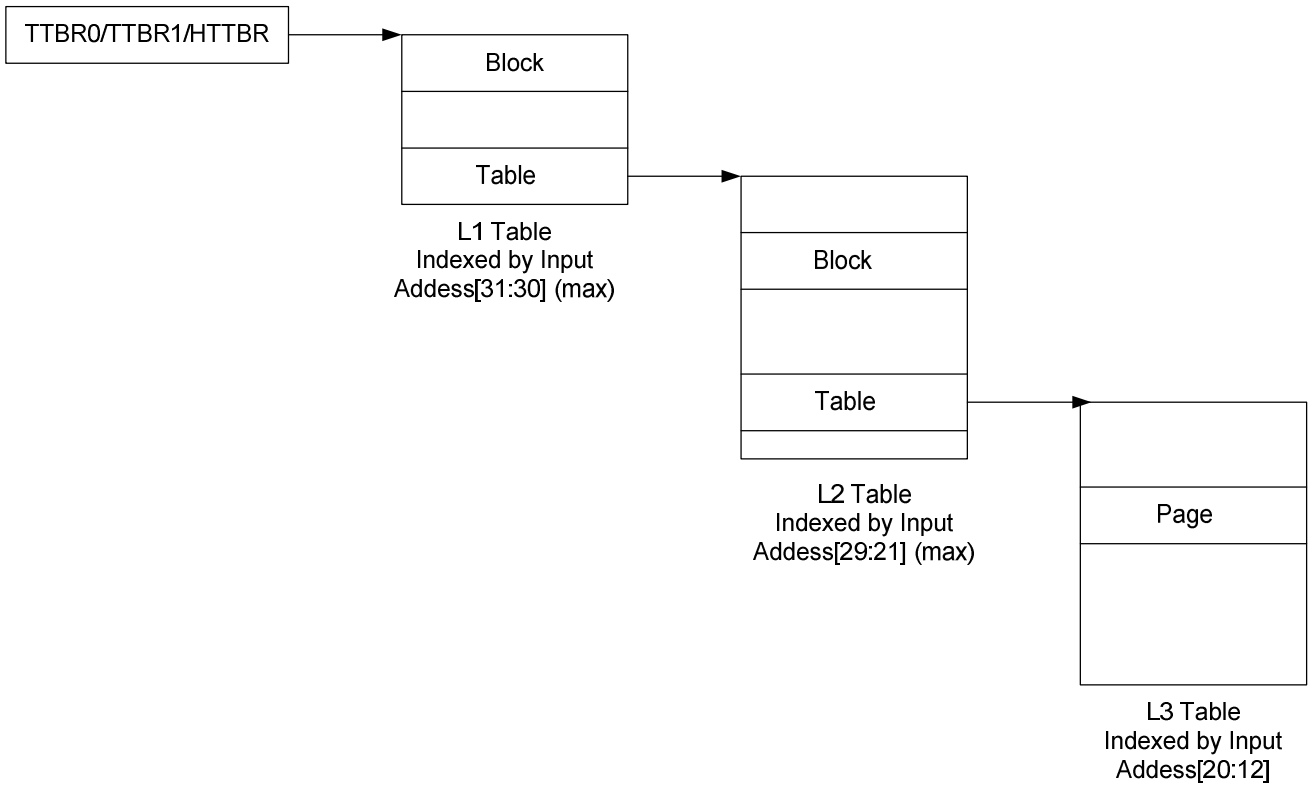
2^N-1 (32 >= N >= 25)

4.2.4.2 Translations in Hyp Mode

Translations in Hyp mode use a single translation base register (HTTBR), which is the equivalent of the TTBR0 for translation of memory accesses made in Hyp mode. The behaviour of the translation system for memory accesses made in Hyp mode is consistent with the HTCR.EPD1 bit being permanently set.

4.2.4.3 Diagrammatic representation of the first stage translation table walk

The following diagram shows the form of the first stage translation table walk



4.2.5 Performing the translation table walk – Second stage of translation

This translation table system can be used for a second stage of translation with a 40-bit Input address (taken from the IPA produced by the first stage of translation). Where the input address of the second stage of translation is a 32 bit address (as would be the case where the translation has been performed using the existing ARMv7 VMSA mechanism) the address is zero extended. The second stage of translation is based on a two or three level table, starting at level 2 or level 1 as appropriate; the number of levels is explicitly described in the translation control register (using the VTCR.SL0 field). The architecture places some restrictions on the relationship between the number of levels of translation table walk and the size of the input address (as described in the VTCR.T0SZ field):

- If the Input address range is less than or equal to 2^{30} bytes, then the starting level is level 2, requiring two levels of tables to provide 4KByte granularity mappings.
- If the Input address range is greater than 2^{30} bytes but less than or equal to 2^{34} bytes, then the starting level can be configured to be either level 1 or 2, requiring either three or two levels of tables to provide 4KByte granularity mappings.
- If the Input address range is greater than 2^{34} bytes, then the starting level is level 1, requiring three levels of tables to provide 4KByte granularity mappings.

Where the Input address range is larger than the size that can be accommodated by the configured number of levels of translation table walk using a single table at the highest level, multiple highest level tables need to be placed contiguously in memory to allow them to be addressed by the same translation table base register without the need to use a higher level table. These multiple tables must be aligned to the total size of the contiguous block of tables.

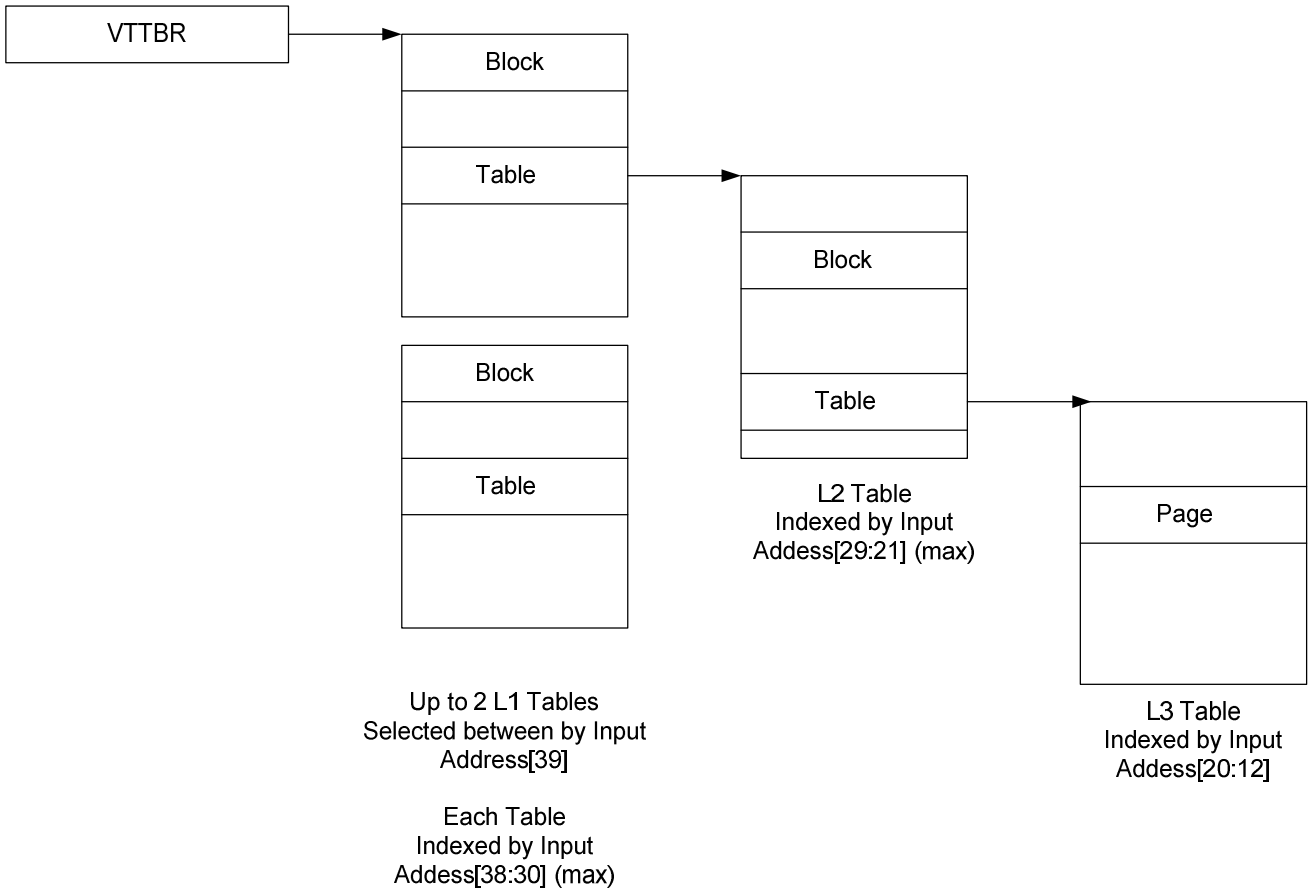
The second stage of translation only provides a single translation base register, covering a Input address range from 0. A field in the corresponding second stage translation base control register indicates the size of the Input address range supported, as an integer power of 2. A translation fault is generated if the Input address is larger than the address range supported.

The maximum Input address translated by the second stage of translation can be configured to be :

2^N-1 (40 >= N >= 25)

4.2.5.1 Diagrammatic representation of the second stage translation table walk

The following diagram shows the form of the second stage translation table walk



4.2.6 Improving the caching of translation entries by providing contiguous hints

The translation table entries contain a bit which is used to provide a hint that 16 adjacent translation table entries (aligned such that the upper 5 bits of the Input address range to index the table are all the same) point to a contiguous Output Address range. The contiguous Output Address range must be aligned to the size described of 16 translation table entries at the same level of translation table.

This hint can then allow the TLB to cache a single entry in the TLB covering those multiple entries. This provides a capability similar to supersections and large pages, in a systematic manner.

This bit acts solely as a hint, and does not mandate the form of caching in the TLB. In particular, any TLB maintenance by address must still be performed assuming that the entries have been cached without relying on this hint

4.2.7 Security status of translation table walks

The ARM Security Extensions tied the physical address map (secure or non-secure) chosen for a translation table walk to the security state of the process that caused the translation table miss. It has been observed that this removes the possibility of holding level 3 translation tables in non-secure memory when they are being accessed by the secure side but contain only non-secure mappings.

This functionality is provided in the Large Physical Address Extensions by adding a bit (NSTable) to the attributes for each translation table to indicate that subsequent levels are to be fetched from either secure or non-secure memory (depending on the value of this bit).

Where a table descriptor has been fetched from non-secure memory, the NSTable bit held in that table descriptor is ignored as the entry can only indicate access to non-secure memory. Where the table descriptor is held in secure memory the NSTable bit held in that table descriptor determines whether the subsequent table access will be to secure or non-secure memory.

Where a block or page descriptor has been fetched from non-secure memory, the NS bit held in that table is ignored as the entry can only indicate access to non-secure memory. Where a block or page descriptor is held in secure memory the NS bit held in that table determines whether the entry can only indicate access to non-secure memory.

In order to prevent TLB entries of “Global” entries from Non-secure memory used for Secure translations (as a result of NSTable being set) overlaying Secure non-Global entries belonging to other entries, all Secure state translations fetched with NSTable set are treated as being non-Global.

4.2.8 First Stage Memory Attribute lookup

The Large Physical Address Extensions provide a revised version of the TEX-Remap system to indirect the selection of memory attributes from the page table entries. This takes the 3 bit AttrIndx field in the translation table entry, and uses this to access an 8 bit field in one of the 32-bit Memory Attribute Indirection Registers, MAIR0, MAIR1.

MAIR0 is selected when AttrIndx[2]==0. MAIR1 is selected when AttrIndx[2]==1.

The field selected in the MAIR* register is determined by AttrIndx[1:0], as

bits[8*AttrIndx[1:0]+7:8*AttrIndx[1:0]]

The encoding of each MAIR is as follows:

Bits[7:4]	**Meaning**
0000	SO or Device Memory
0100	Normal Memory, Outer Non-Cacheable
10RW	Normal Memory, Outer Write-through
11RW	Normal Memory, Outer Write-back

R= Outer Read Allocate Policy; W=Outer Write Allocate Policy

All other values UNPREDICTABLE.

Bits[3:0]	**Meaning when Bits[7:4] == ‘0000’**	**Meaning when Bits[7:4] != ‘0000’**
0000	Strongly Ordered	UNPREDICTABLE
0100	Device	Normal Memory Inner Non-Cacheable
10RW	UNPREDICTABLE	Normal Memory Inner Write-through
11RW	UNPREDICTABLE	Normal Memory Inner Write-back

R= Inner Read Allocate Policy; W= Inner Write Allocate Policy

The allocation policy is encoded as: 0 – Don’t allocate, 1- Allocate.

All other values UNPREDICTABLE

The MAIR* registers replace the PRRR and NMRR registers in the existing VMSAv7 architecture, with MAIR0 having the same register encoding as the PRRR, and MAIR1 having the same register encoding as the NMRR.

4.2.9 Introduction of a complete set of Cache Allocation hints

VMSAv7 has a set of cache allocation hints which can be applied for the cacheable normal memory types to allow software to communicate to the hardware whether it is desirable to allocate into the cache. These hints are incomplete in the VMSAv7, in that some options, such as the ability to determine whether a WT location is Write-Allocate or not were not included for encoding reasons.

In addition, there are some circumstances where the ability to communicate that a memory location is WB or WT cacheable, but that if the line is not present in a cache, then it is not beneficial to allocate the location into the cache. This gives the concept that a line might be Write-Back or Write-Through Cacheable, but have neither Read nor Write Allocation.

Note: Cacheable with neither Read nor Write Allocation is not the same as Non-Cacheable. Non-Cacheable locations have guarantees of coherency with respect to observers outside the Shareability domains that do not apply to Cacheable locations with neither Read nor Write Allocation

The LPAE introduces a full set of allocation hints for both WT and WB cacheable locations. As with the VMSAv7 definition, how these locations are handled in an implementation is outside the scope of the architecture. Unlike the Shareability, Memory Type and Cacheability attributes, there is no requirement that different aliases of a memory location have the same allocation hints.

4.2.10 Domains

The translation table format introduced by the Large Physical Address Extensions does not support domains. All addresses are treated as if they are within the Client Domain, and the DACR has no effect.

4.2.11 PXN – Privileged XN

In the ARMv7 VMSA, the XN permission check applies to all instruction fetches, regardless of the privilege of the instruction fetch. The Large Physical Address Extensions add a new concept, Privileged XN, which prevents Privileged Execution of a region of memory. The behaviour of Privileged XN is equivalent to that of XN, but applies only to instructions fetched from Privileged modes.

Privileged XN does not change the functionality of XN, which is to prevent any execution of the memory location.

4.2.11.1 Addition of Privileged XN to the existing VMSAv7 translation table format

In an implementation of the Large Physical Address Extensions, the Privileged XN capability can be defined for regions of memory at 1MByte granularity, in either the Section/Supersection or the Page Table descriptors of a level 1 page table.

- For Section/Supersection first level descriptors, bit[0] of the descriptor is used to encode the PXN bit. This means that the descriptor entry with bits[1:0] = ‘11’ is no longer Reserved, but is extended to cover Section/Supersection descriptors with PXN=’1’.
- For Page Table first level descriptors, bit[2] of the descriptor (which is currently SBZ) is used to encode the PXN bit.

In both these cases, the changes are backwards compatible with the VMSAv7.

4.2.12 Hierarchical Read/Write, User/Privileged, XN and PXN override bits in the higher levels of table

The Large Physical Address Extensions provide a mechanism where an entry in the translation tables at one level sets limits on the access permissions for all translations made by entries in the subsequent tables within the same stage of translation. This allows the Kernel to re-use a set of mappings for an application, but to override the User Accessibility, or Writeability, or Executability of those pages when the mappings are being re-used.

This overriding of the permissions uses the table permission bits of the translation table entries, and the permission modification requires that permission for a particular type of access must be granted at each level of the table. Therefore this feature has no effect in table descriptors whose entries have:

- APTable[1:0]=00
- XNTable = 0

- PXNTable = 0

The encodings of APTable are:

APTable[1]	APTable[0]	Effect
0	0	No affect on the permissions in subsequent tables in the same stage of translation
0	1	User access is not permitted
1	0	Write access (either User or Privileged) is not permitted
1	1	User access is not permitted Privileged Write access is not permitted

Mechanism is only provided in stage 1 translation – for stage 2 the relevant fields are SBZ.

Note: The Hierarchical attributes act as overrides to the final permission of a valid translation table entry, and are not permissions for table entries in their own right.

The effect of hierarchical attributes might be cached in multiple TLB entries, being any entries covering translation table entries that are affected by the hierarchical attributes. Therefore changing the hierarchical attributes can require some coarse grained invalidation of the TLB in order to ensure that the effect of the change is visible to subsequent memory transactions.

4.2.13 Single Copy Atomicity of LDRD/STRD

The Large Physical Address Extensions architecture requires that LDRD and STRD accesses to 64-bit aligned locations are 64-bit single-copy atomic as seen by translation table walks and accesses to translation tables.

Note: The determination of atomicity of memory transactions depends on properties of system memory architecture that is outside the scope of the processor architecture..

4.2.14 Supersections and the LPAE

The VMSAv7 contains a mechanism to allow access to a 40-bit physical address space through the use of Supersections. In an implementation that supports both the LPAE and VMSAv7, if the implementation supports more than 32 bits of Physical Address, it is required that Supersections provide a mechanism to access that address space as well.

4.2.15 Handling of the ASID in the LPAE

In the ARMv7 VMSA, the ASID is held in the bottom 8 bits of the ContextID Register, and so is not updated atomically with the changes to the TTBR0 or TTBR1. This gives rise to a complexity when changing the TTBRx and the ASID in ensuring that entries in the TLB, or other caching structures, are not associated with the wrong ASID value. Incorrect tagging could give rise to errors when the cached entries are used. Strategies to avoid this are outlined in [1, section B3.10.5]. The possibility associated with incorrect tagging increases with the increasing use of speculative memory accesses.

With the LPAE, the larger descriptor for the translation table base register gives rise to an alternative solution to this problem by holding the ASID and the Translation Table Base Address in the same register. As there are two different Translation Table Base Address registers, the operating system is given the ability to configure in the TTBCR which of the two fields is used as the ASID; the other field is ignored by hardware This means that one Translation Table Base Address and the ASID can be changed atomically, and as a result, no special measures, such as those outlined in [1, section B3.10.5] are needed in any of the following cases:

- When TTBR0 is the only Translation Table Base that is being used
- When the only translation tables that hold non-global entries are pointed to by TTBR0.
- When the only translation tables that hold non-global entries are pointed to by TTBR1

If the operating system uses two translation tables and both tables hold non-global entries, one of the approaches described in [1, section B3.10.5] must be employed to achieve a change of the ASID and both TTBR registers to ensure that entries in the TLB, or other caching structures, are not associated with the wrong ASID value .

When the LPAE is enabled, the ASID field in the ContextID register has no significance for the memory translation system, and is merely an extension of the ContextID register.

4.2.16 Coherency Requirements for Cacheable Translation Table Walks

When the LPAE is enabled, all cacheable Translation Table Walks are required to look in all levels of cache within the shareability domain described by the processor. This requirement is the same as the requirement for translation table walks introduced as part of the MP Extensions, but also applies in an implementation that does not include the MP Extensions.

4.2.17 Pseudo-code for the translation table walk

The algorithm for the translation table walk is as described in the following pseudo-code.

```
BaseAddress<39:0> = Zeros(40);
BaseFound = FALSE;
Disabled = FALSE;
if Stage1Translation then
    if InHypMode() then
        LookupSecure = FALSE;
        T0Size = UInt(HTCR.T0SZ);
        if T0Size == 0  || IsZero(IA<31:(32-T0Size)>) then
            CurrentLevel = (HTCR.T0SZ<2:1>==’00’) ? 1: 2;
            BALowerBound = 9*CurrentLevel – T0Size - 4;
            BaseAddress<39:0> = HTTBR<39:BALowerBound>:Zeros(BALowerBound);
            if IsZero(HTTBR<BALowerBound-1:3>) == FALSE then UNPREDICTABLE;
            BaseFound = TRUE;
            StartBit = 31-T0Size;

    else

        LookupSecure = IsSecure();
        T0Size = UInt(TTBCR.T0SZ);
        if T0Size == 0  || IsZero(IA<31:(32-T0Size)>) then
            CurrentLevel = (TTBCR.T0SZ<2:1>==’00’) ? 1: 2;
            BALowerBound = 9*CurrentLevel – T0Size - 4;
            BaseAddress<39:0> = TTBR0<39:BALowerBound>:Zeros(BALowerBound);
            if IsZero(TTBR0<BALowerBound-1:3>) == FALSE then UNPREDICTABLE;
            BaseFound = TRUE;
            Disabled = (TTBCR.EPD0 == ’1’);
            StartBit = 31-T0Size;

        T1Size = UInt(TTBCR.T1SZ);
        if (T1Size == 0 && !BaseFound) || IsOnes(IA<31:(32-T1Size)>) then
            CurrentLevel = (TTBCR.T1SZ<2:1>==’00’) ? 1: 2;
            BALowerBound = 9*CurrentLevel – T1Size - 4;
            BaseAddress<39:0> = TTBR1<39:BALowerBound>:Zeros(BALowerBound);
            if IsZero(TTBR1<BALowerBound-1:3>) == FALSE then UNPREDICTABLE;
            BaseFound = TRUE;
            Disabled = (TTBCR.EPD1 == ’1’);
            StartBit = 31-T1Size;

else
    T0Size = SInt(VTCR.T0SZ);
    SLevel = UInt(VTCR.SL0);
    BALowerBound = 14 – T0Size - 9*SLevel;
```

```
    // check UNPREDICTABLE combinations of the Starting level and Size fields
    // and check the VTTBR is aligned correctly
    if SLevel == 0 && VTCR.T0SZ < -2 then UNPREDICTABLE;
    if SLevel == 1 && VTCR.T0SZ >  1  then UNPREDICTABLE;
    if VTCR.SL0<1> == '1' then UNPREDICTABLE;
    if IsZero(VTTBR<BALowerBound-1:3>) == FALSE then UNPREDICTABLE;

    // zero extend the address to 40 bits if necessary
    if FirstStageTranslationWas32Bits() then
        IA<39:32> = '00000000';

    if T0Size == -8 || IsZero(IA<39:(32-T0Size)>) then
        CurrentLevel = 2-SLevel;
        BaseAddress<39:0> = VTTBR<39:BALowerBound>:Zeros(BALowerBound);
        BaseFound = TRUE;
        StartBit = 31-T0Size;
    LookupSecure = FALSE;

if !BaseFound || Disabled then
    TranslationFault()

FirstIteration = TRUE;
TableRW = TRUE;
TableUser = TRUE;
TableXN = FALSE;
TablePXN = FALSE;

repeat

    LookUpFinished = TRUE;
    BlockTranslate = FALSE;
    Offset = 9*CurrentLevel;
    if FirstIteration then
        IASelect = ZeroExtend(IA<StartBit:39-Offset>:'000', 40);
    else
        IASelect = ZeroExtend(IA<47-Offset:39-Offset>:'000', 40);
    LookupAddress = BaseAddress OR IASelect;

    FirstIteration = FALSE;

    // If there are two stages of translation, then the first stage table walk addresses
    // are themselves subject to translation
    if LastStageOfTranslation() then
        Descriptor = _Mem[LookupAddress,LookupSecure,8];
    else
        LookupAddress = SecondStageTranslate(LookupAddress);
        Descriptor = _Mem[LookupAddress, FALSE, 8] ;

    if Descriptor<0> == '0' then
        TranslationFault(CurrentLevel);
    else
        if Descriptor<1> == '0' then
           if CurrentLevel == 3 then
               TranslationFault(CurrentLevel);
           else
               BlockTranslate = TRUE;
        else
```

```
            if CurrentLevel == 3 then
                BlockTranslate = TRUE;
            else // table translation
                BaseAddress = Descriptor<39:12>:’000000000000’;
                LookupSecure = LookupSecure && (Descriptor<63> == ’0’);
                TableRW = TableRW && (Descriptor<62> == ’0’);
                TableUser = TableUser && (Descriptor<61> == ‘0’);
                TablePXN = TablePXN || (Descriptor<59> == ‘1’);
                TableXN = TableXN || (Descriptor<60> == ‘1’);
                LookUpFinished = FALSE;

    if BlockTranslate then
        OutputAddress = Descriptor<39:39-Offset> : IA<38-Offset:0>;
        Attrs = Descriptor<54:52>: Descriptor<11:2>;

        if Stage1Translation then
            if TableXN then Attrs<12> = ‘1’;
            if TablePXN then Attrs<11> = ‘1’;
            if IsSecure() && !(LookupSecure) then Attrs<9> = ‘1’;
            if !(TableRW) then Attrs<5> = ‘1’;
            if !(TableUser) then Attrs<4> = ‘0’;
            if !(LookupSecure) then Attrs<3> = ‘1’;
    else
        CurrentLevel = CurrentLevel + 1;
until LookUpFinished
// final Attrs<> bus contains:
// 12:    XN
// 11:    PXN
// 10:    Contiguous Hit
// 9:     nG
// 8:     AccessFlag
// 7:6:   Shareability
// 5:     1: ReadOnly 0: Read/Write
// 4:     1: User 0: Privileged only
// 3:0:   Stage2: Memory Type
// 3:     Stage1: Non-Secure
// 2:0:   Stage1: Memory Type Index
```

4.2.17.1 Additional Pseudo-code Functions

The pseudo-code for the translation table walk includes an addition pseudo-code function:

```
SecondStageTranslate(Address)
```

This function performs a second stage translation, from IPA to PA, of the supplied address. The second stage translation might hit in a TLB, or might involve a translation table walk using the algorithm described in section 4.2.17.

4.3 CP15 Register Definitions

4.3.1 General Behaviours

4.3.1.1 Reset

The registers in this section are all UNKNOWN at reset, unless otherwise stated.

4.3.1.2 UNK/SBZP and UNK/SBOP behaviours

The new translation table format re-uses some registers that were allocated into the ARMv7 VMSA, using the TTBCR.EAE bit to distinguish which format is being used. In some cases, the new translation table format allocates functionality to fields or bits that were UNK/SBZP or UNK/SBOP in the ARMv7 VMSA format, or vice versa. For such fields, unless otherwise stated in this specification, the hardware treatment of UNK/SBZP or UNK/SBOP is modified from the behaviour described in [1], which requires that such bits return 0 (for UNK/SBZP) or 1 (for UNK/SBOP) and ignore writes.

For these fields and bits, when the setting of the TTBCR.EAE bit is such that the field or bit is described as UNK/SBZP or UNK/SBOP then:

- Reads must return the value that was last written to that field or bit in the register, regardless of the value of the value of the TTBCR.EAE when it was written. If it has not been written since reset, then it will return the specified reset value, if there is one, or an UNKNOWN value.
- Writes must cause an update to the storage location associated with that field or bit.
- The state of the storage location associate with a field or bit must have no other effect on the behaviour of the processor other than determining the value to be read back while the setting of the TTBCR.EAE bit is such that the field or bit is described as UNK/SBZP or UNK/SBOP.

If the TTBCR.EAE bit is changed such that the field or bit changes from being described as UNK/SBZP or UNK/SBOP to having an allocated function, the value that was last written to the field or bit has an effect on the processor.

These revised behaviours only apply to fields which are UNK/SBZP or UNK/SBOP in one state of TTBCR.EAE, and have an allocated meaning in the other state. If a bit is UNK/SBZP or UNK/SBOP regardless of the value of the TTBCR.EAE bit, the behaviours are unchanged from that described in [1].

These revised behaviours only apply to the hardware treatment of the fields or bits. The software behaviour remains unchanged, in that the fields should be preserved if appropriate or written to the value required by the “Should Be” if being written to a new value, or else the effect is UNPREDICTABLE.

Implementation note: the purpose of these rules is to require that such fields or bits are implemented as flip-flops whose interpretation is changed by the value of the TTBCR.EAE bit, and which can be written or read regardless of the setting of the TTBCR.EAE bit.

4.3.2 TTBR0/TTBR1/HTTBR and VTTBR

New formats are required for the stage 1 TTBR0 and TTBR1, compared with the ARMv7 VMSA system – most notably they extend to become 64-bit wide registers in order to be able to hold the wider base Output addresses. In order to handle small tables, the non-address fields that are present in the ARMv7 VMSA TTBRx registers in bits [6:0] are held in the TTBCR register when the new format is in use.

It is necessary that TTBR0, TTBR1, HTTBR and VTTBR registers are updated atomically by a single instruction (MCRR based instructions).

For the stage 1 translation except in Hyp mode, this constitutes an extension from 32 bits to 64 bits of the TTBR0 and TTBR1 registers, rather than the introduction of new separate registers. The TTBCR.EAE bit selects whether the old encoding or the new encoding is used for translation table walks. The existing ARMv7 VMSA mechanism will access bits[31:0] of the extended TTBR0 or TTBR1.

When the TTBCR.EAE bit is clear, then the translation table walk uses the bottom 32 bits of the TTBRx register using the format described in the VMSAv7 [1], and bits[63:32] of the TTBRx are IGNORED. When TTBCR.EAE is set, then the translation table walk interprets the 64 bits of the TTBRx register as shown below.

This allows a hypervisor to access the entire TTBR0 or TTBR1 using 64-bit accessing instructions regardless of whether the Virtual Machine is using a 32-bit or a 40-bit Output address.

The layout of the TTBR0, TTBR1, HTTBR and VTTBR registers when using the new translation table format is as follows:

UNK/SBZP	TTBR0/1: ASID[7:0] HTTBR: UNK/SBZP VTTBR: VMID[7:0]	UNK/ SBZP	BADDR[39:x]	UNK/SBZP

BADDR (Bits[39:x]): Translation Base Address[39:x]

ASID/VMID (Bits[55:48]): ASID/VMID associated with this base address. The selection between the TTBR0 ASID and the TTBR1 ASID filed is determined by the TTBCR.A1 field (see section 4.2.15)

The VMID field of the VTTBR is reset to 0.

For TTBR0/1, the value x is determined by TTBCR.T0/1SZ using the formula:

```
TxSize = UInt(TTBCR.T*SZ);
if TxSize  > 1 then
    x = 14 – TxSize;
else
    x = 5 – TxSize;
```

Similarly, for HTTBR, the value x is determined by HTCR.T0SZ in the same way:

```
T0Size = UInt(HTCR.T0SZ);
if T0Size  > 1 then
    x = 14 – T0Size;
else
    x = 5 – T0Size;
```

The stage 1 TTBR0, TTBR1 and HTTBR registers are accessed as a 64-bit quantity using:

MRRC/MCRR p15, 0, Rt1, Rt2, c2:	TTBR0	Banked
MRRC/MCRR p15, 1, Rt1, Rt2, c2:	TTBR1	Banked
MRRC/MCRR p15, 4, Rt1, Rt2, c2:	HTTBR	Banked-NSHyp only

In all these cases, Rt1 contains the bottom 32 bits being transferred, while Rt2 contains the top 32 bits.

Stage 2 supports the VTTBR, which has the same format as the revised TTBR0.

For VTTBR, the value x is determined by VTCR.{T0SZ,SL0} using the formula:

```
T0Size = SInt(VTCR.T0SZ);
if VTCR.SL0 == ’00’ then
   x = 14 – T0Size;
else
   x = 5 – T0Size;
```

The stage 2 VTTBR register is accessed as a 64-bit quantity using:

MRRC/MCRR p15, 6, Rt1, Rt2, c2:	VTTBR	Banked-NSHyp only

Rt1 contains the bottom 32 bits being transferred, while Rt2 contains the top 32 bits.

The secure copy of the TTBR0 is affected by the CP15SDisable input.

4.3.3 TTBCR/HTCR and VTCR

The TTBCR and HTCR, which control stage1 of the translation, are significantly different from the VTCR, which controls stage 2 of the translation.

4.3.3.1 Stage 1

The revised layout of the TTBCR and HTCR is:

31	30	29	28	27	26	25	24	23	22	21	19	18	16	15	14	13	12	11	10	9	8	7	6	3	2	0
EAE	IMP	SH1		ORGN1		IRGN1		EPD1	A1	(000)		T1SZ		(00)		SH0		ORGN0		IRGN0		EPD0	(0000)		T0SZ	

EAE (Bit[31]): Extended Address Enable

0: Use the 32-bit Translation System in the ARMv7 VMSA

1: Use the 40-bit Translation System described in this document

For the HTCR, this bit is UNK/SBOP

For the Secure and Non-secure copies of the TTBCR, this bit is reset to 0.

The remaining bits are assigned if EAE is 1 – if EAE is 0, then the format in the ARMv7 VMSA is used.

IMP (Bit[30]): IMPLEMENTATION DEFINED

SH1(Bits[29:28]): Shareability attributes for the memory associated with the translation table walks using TTBR1. The SH1 field is encoded in the same format as in section 4.2.3

For the HTCR, this field is UNK/SBZP.

ORGN1(Bits[27:26]): Outer Cacheability attributes for the memory associated with the translation table walks using TTBR1

For the HTCR, this field is UNK/SBZP. The ORGN1 field is encoded in the same format as in the RGN of the TTBR1 described in [1].

IRGN1(Bits[25:24]): Inner Cacheability attributes for the memory associated with the translation table walks using TTBR1. The IRGN1 field is encoded in the same format as in the IRGN of the TTBR1 described in [1].

For the HTCR, this field is UNK/SBZP.

EPD1 (bit[23]): Translation Walk Disable for TTBR1 region for EAE==1.

This bit controls whether a translation table walk is performed on a TLB miss when TTBR1 is used:

0: If a TLB miss occurs when TTBR1 is used a translation table walk is performed.

1 If a TLB miss occurs when TTBR1 is used no translation table walk is performed and a L1 Translation fault is returned.

Note: This function is the same as PD1; the bit position is moved depending on the setting of EAE

For the HTCR, this bit is UNK/SBOP

A1 (Bit[22]: Select ASID from TTBR1 ASID field. If this bit is 0, then the ASID is selected from the TTBR0 ASID field.

For the HTCR, this bit is UNK/SBZP

T1SZ (Bits[18:16]): The Size offset of the TTBR1 addressed region, encoded as a 3 bit unsigned number, giving the size of the region as $2^{32-T1SZ}$

For the HTCR, this field is UNK/SBZP

SH0(Bits[13:12]): Shareability attributes for the memory associated with the translation table walks using TTBR0/HTTBR as relevant The SH0 field is encoded in the same format as in section 4.2.3

ORGN0(Bits[11:10]): Outer Cacheability attributes for the memory associated with the translation table walks using TTBR0/HTTBR as relevant. The ORGN0 field is encoded in the same format as in the ORGN of the TTBR0 described in [1].

IRGN0(Bits[9:8]): Inner Cacheability attributes for the memory associated with the translation table walks using TTBR0/HTTBR as relevant. The IRGN0 field is encoded in the same format as in the ORGN of the TTBR0 described in [1].

EPD0 (bit[7]): Translation Walk Disable for TTBR0 region for EAE==1.

For the HTCR, this field is UNK/SBZP.

This bit controls whether a translation table walk is performed on a TLB miss when TTBR0 is used:

0 If a TLB miss occurs when TTBR0 is used a translation table walk is performed.

1 If a TLB miss occurs when TTBR0 is used no translation table walk is performed and a L1Translation fault is returned.

Note: This function is the same as PD0; the bit position is moved depending on the setting of EAE

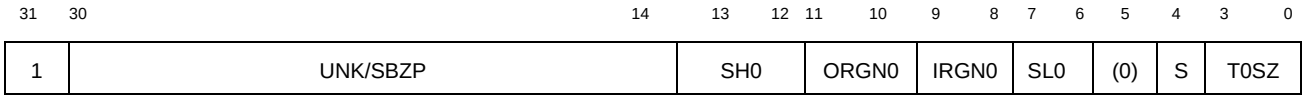
T0SZ (Bits[2:0]): The Size offset of the TTBR0/HTTBR addressed region, encoded as a 3 bit unsigned number giving the size of the region as $2^{32-T0SZ}$

The secure copy of the TTBCR register is reset to 0, and the Non-secure TTBCR.EAE bit is reset to 0.

The secure copy of the TTBCR is affected by the CP15SDisable input.

4.3.3.2 Stage 2

The layout of the VTCR is:



SH0(Bits[13:12]): Shareability attributes for the memory associated with the translation table walks using VTTBR

ORGN0(Bits[11:10]): Outer Cacheability attributes for the memory associated with the translation table walks using VTTBR

IRGN0(Bits[9:8]): Inner Cacheability attributes for the memory associated with the translation table walks VTTBR

SL0 (Bits[7:6]): Starting Level for VTCR addressed region:

0: Level2

1: Level1

2: UNPREDICTABLE

3: UNPREDICTABLE

S (bit[4]) This bit must be programmed to T0SZ[3], or the effect is UNPREDICTABLE

Note: This bit is a sign-extension of the T0SZ field, and is allocated this way for future compatibility for translation table systems with a larger input address

T0SZ (Bits[3:0]): The Size offset of the VTCR addressed region, encoded as a 4 bit signed number giving the size of the region as $2^{32-T0SZ}$

4.3.4 Fault Status registers

When the ARMv7 VMSA translation table format is selected on an implementation with the Large Physical Address Extensions, the Fault Status register information is unchanged from that described as part of ARMv7 VMSA, with two exceptions for the DFSR:

1. For Permission Faults that cause a Data Abort, the Domain field (DFSR[7:4]) is UNKNOWN.
2. A new bit (CM, bit[13]) is added to indicate that the data fault came from a Cache Maintenance Operation. This bit is valid for synchronous faults only.

When the new translation table format is selected and an abort is taken in Abort or Monitor modes in the same security state that it was generated in, the Fault Status encoding is reworked as part of the Large Physical Address Extensions to give a cleaner mapping of the fault information as follows:

DFSR and IFSR: bit[10] and bits[7:6] are UNK/SBZP. Bits[5:0] encode the faults in a new format shown below

In addition, for aborts taken in Hyp mode, and when in Debug state, for aborts that would be taken in Hyp mode had the processor not been in Debug state, the fault status code using the new format is held in bits [5:0] of the HSR.

Note: In Debug state, the processor does not change mode on taking an Abort.

If an External Abort from the non-secure state is taken in Monitor mode, the DFSR or IFSR format is the new format if any of the following apply:

- The Secure state TTBCR.EAE is set
- The External Abort is Synchronous from Hyp mode
- The External Abort is Synchronous and is from a mode other than Hyp mode, and the Non-secure TTBCR.EAE bit is set.

Fault Status [5:0]	Meaning
0001LL	Translation Fault – level determined by LL bits
0010LL	Access Flag fault – level determined by LL bits
0011LL	Permission fault – level determined by LL bits
010000	Synchronous External Abort
011000	Memory access Synchronous Parity error
010001	Asynchronous External Abort (Data code only)
011001	Memory access Asynchronous Parity error (Data code only)
0101LL	Synchronous External Abort on Translation Table walk – level determined by LL bits
0111LL	Memory access Synchronous Parity error on Translation Table walk – level determined by LL bits
100001	Alignment Fault
100010	Debug event
110100	IMPLEMENTATION FAULT (Lockdown Abort) (Data code only)
111010	IMPLEMENTATION FAULT (Coprocessor Abort) (Data code only)

If a Data Abort exception is generated by an instruction cache maintenance operation when the new translation table format is selected, the fault is reported as a Cache Maintenance fault in the DFSR (or HSR) with the appropriate Fault Status code. For such exceptions reported in the DFSR, the corresponding IFSR is UNKNOWN.

The LL fields encode which table level the fault is associated with:

Bit[1]	Bit[0]	
0	0	Reserved
0	1	Level1
1	0	Level2
1	1	Level3

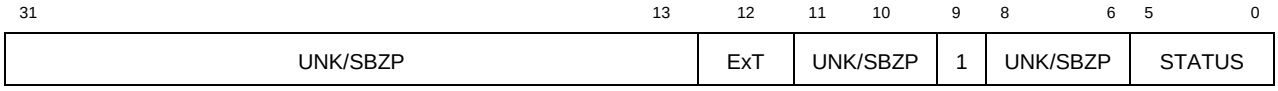
The level that a fault is associated with is:

- For faults associated with a translation table walk, the level of table walk being performed
- For translation faults, the level of translation table that gave the fault. If the fault was caused by a disabled translation table walk or if the size of the address presented is out of the range specified for matching with any base address register, the fault is reported for Level1.
- For access faults, the level of translation table that gave the fault.
- For permission faults, including those caused by hierarchical permissions, it is the final level of translation table used for that translation (ie the level of that gave block entry, or Level 3 if the other levels didn't give a block entry).

The overall layout of the DFSR and IFSR when the new translation table format is selected is as shown below:

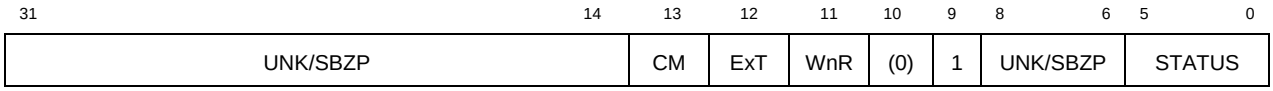
IFSR

The layout of the IFSR when the new translation table format is selected is:



DFSR

The layout of the DFSR when the new translation table format is selected is:



Bit[9] is a status bit allows software to distinguish between the old format and the new format registers without having to refer to the TTBCR.EAE bit. The bit is not interpreted by the hardware to determine the behaviour of the memory system and can be written to any value by software, and will return the last value written unless the register has been updated as a result of a fault. Bit[9] in the VMSAv7 DFSR and IFSR formats is defined to be 0 from the inclusion of the LPAE.

4.3.5 Fault Address registers

The Virtualization Extensions define that the HIFAR/HDFAR always hold the VA, and so there no need to expand these registers.

Note: This differs from earlier draft versions of this specification.

4.3.6 Physical Address Register

The Physical Address Register (PAR) is extended to 64 bits to hold a 40-bit Output address, and becomes a 64-bit register. This constitutes an extension of the PAR, rather than the introduction of a new separate register. The existing ARMv7 VMSA mechanism will access bits[31:0] of the extended PAR.

The PAR registers are accessed as a 64-bit quantity using:

MRRC/MCRR p15, 0, Rt1, Rt2, c7: PAR Banked

Rt1 contains the bottom 32 bits being transferred, while Rt2 contains the top 32 bits.

4.3.6.1 Encoding of the PAR

The PAR is recoded when the Extended Address is enabled, to allow the additional features of the Large Physical Address Extensions to be captured in the PAR. When the Extended Address is not enabled, PAR[63:32] are UNK/SBZP

PAR register contents on a Fault

When the new translation table format is select, if a CP15 VA to PA translation operation is performed that causes an abort, the PAR register holds the following values:

PAR[63:12]:	UNK/SBZP
PAR[11]:	1
PAR[10]:	UNK/SBZP
PAR[9]:	0: Fault in First Stage of Translation
	1: Fault in Second Stage of Translation
PAR[8]:	Indicates that the fault was from a second stage fault during a first stage translation table walk
PAR[7]:	UNK/SBZP
PAR[6:1]:	Fault Status code shown in section 4.3.4
PAR[0]:	1, indicating a Fault (this is the F bit)

PAR register contents when there is not a Fault

When the new translation table format is selected, if a CP15 VA to PA translation operation is performed that does not cause an abort, the PAR register holds the following values:

PAR[63:56]:	Memory Attributes, following the encoding in section 4.3.7 for the MAIR field
PAR[55:40]:	UNK/SBZP
PAR[39:12]:	Physical Address[39:12]
PAR[11]:	1
PAR[10]:	IMPLEMENTATION DEFINED
PAR[9]:	NS
PAR[8:7]:	Shareability, SH[1:0] - this take the value 10 for Device and Strongly-ordered memory
PAR[6:1]:	UNK/SBZP
PAR[0]:	0 – indicating no fault (this is the F bit).

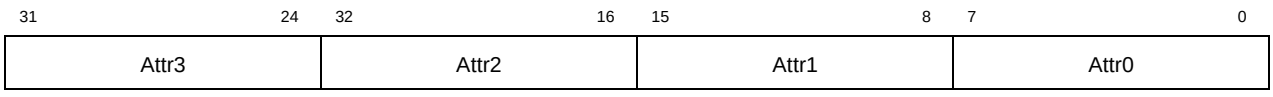
PAR[11]

Bit[11] is a status bit allows software to distinguish between the old format and the new format registers without having to refer to the TTBCR.EAE bit. The bit is not interpreted by the hardware to determine the behaviour of the memory system and can be written to any value by software and will return the last value written unless the register has been updated as a result of an address translation operation. Bit[11] in the VMSAv7 PAR format is defined to be 0 from the inclusion of the LPAE.

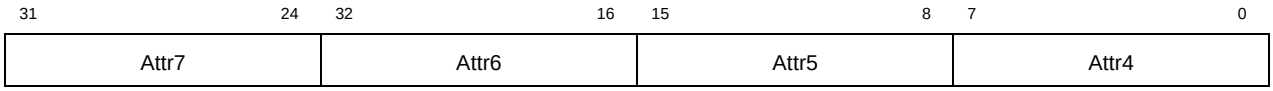
4.3.7 MAIR0/HMAIR0 and MAIR1/HMAIR1

The MAIR0 and MAIR1 registers replace the PRRR and NMRR registers respectively. The HMAIR0 and HMAIR1 provide the same functionality for the first stage of translation used from Hyp mode. They have the same layout.

The MAIR0/HMAIR0 layout is shown in the following diagram



The MAIR1/HMAIR1 layout is shown in the following diagram



In each case, AttrX contains the attribute set described in section 4.2.8 for AttrIndx X, selected in the stage 1 translation tables.

In addition, the CP15 registers addressed by:

MCR p15, 0, Rx, c10, c3, {0,1}

Are reserved for IMPLEMENTATION DEFINED application of attributes associated with the memory encodings. Such attributes are only additional qualifiers to the memory locations, and cannot change the architected behaviours of the memory attributes defined in the MAIR0 and MAIR1 registers.

The secure copies of the MAIR0 and MAIR1 are affected by the CP15SDISABLE input.

4.3.8 Cache and TLB maintenance operations

TLB operations associated with the first stage of translation are unchanged from the VMSAv7, as described in [1]. The Virtualization Extensions add additional TLB maintenance capabilities as described in [2].

The Cache maintenance operations are unchanged from [1].

4.3.9 Debug ROM Address Register, DBGDRAR

The Debug ROM address register (DBGDRAR) is extended to 64 bits to hold a 40-bit physical address, and so becomes a 64-bit register. This constitutes an extension of the DBGDRAR, rather than the introduction of a new separate register. The existing ARMv7 VMSA mechanism will access bits[31:0] of the extended DBGDRAR.

The DBGDRAR registers is accessed as a 64-bit quantity using:

MRRC p14, 0, Rt1, Rt2, c1: DBGDRAR Common

Rt1 contains the bottom 32 bits being transferred, while Rt2 contains the top 32 bits.

The DBGDRAR is extended by the LPAE so that bits[39:12] hold ROMADDR[39:12].

4.3.10 Debug Self-address Offset register, DBGDSAR

The Debug Self-address offset register (DBGDSAR) is extended to 64 bits to hold a 40-bit offset, and becomes a 64-bit register. This constitutes an extension of the DBGDSAR, rather than the introduction of a new separate register. The existing ARMv7 VMSA mechanism will access bits[31:0] of the extended DBGDSAR.

The DBGDSAR registers is accessed as a 64-bit quantity using:

MRRC p14, 0, Rt1, Rt2, c2: DBGDSAR Common

Rt1 contains the bottom 32 bits being transferred, while Rt2 contains the top 32 bits.

The DBGDRAR is extended by the LPAE so that bits[39:12] hold SELOFFSET[39:12], and the upper bits[63:40] are treated as a sign extension of this field.

5 IDENTIFICATION MECHANISMS

5.1 CPU ID

5.1.1 ID_MMFR3

A new field is allocated:

ID_MMFR3[27:24]: Size of Physical Memory Supported by the processor caches

Permitted values are:

0b0000	4GByte (32 bits)
0b0001	64 GBytes (36 bits)
0b0010	1TByte (40 bits)

5.1.2 ID_MMFR0

Two new values are added VMSA support, bits[3:0]

0b0100	As 0b0011 with support for PXN within the first level descriptors
0b0101	As 0b0100 with support for 64-bit translation descriptors